MongoDB, Mongoose and Cloud Storage

Frank Walsh, Diarmuid O’Connor

Agenda

- Cloud Databases
- MongoDB
- Mongoose
- Mongo in the cloud

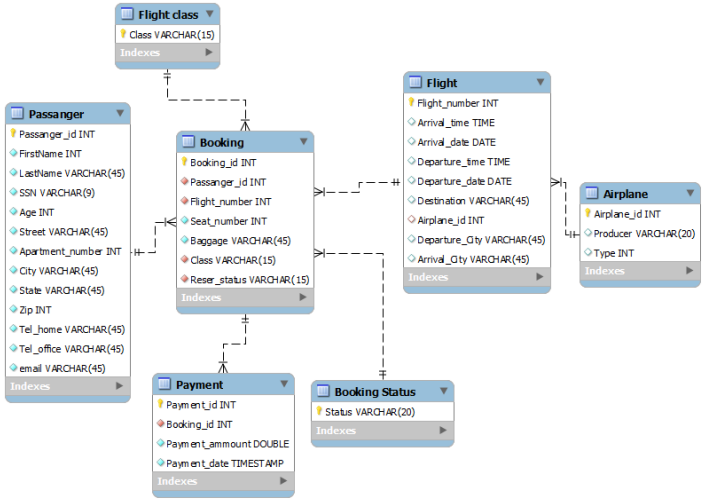
Databases in Enterprise Apps

- Most data driven enterprise applications need a database
  - Persistence: storage of data
  - Concurrency: many applications sharing the data at once.
  - Integration: multiple systems using the same DB
- Enterprise Application DBs require backups, fail over, maintenance, capacity provisioning.
  - Typically handled by a Database Administrator (the DBA).

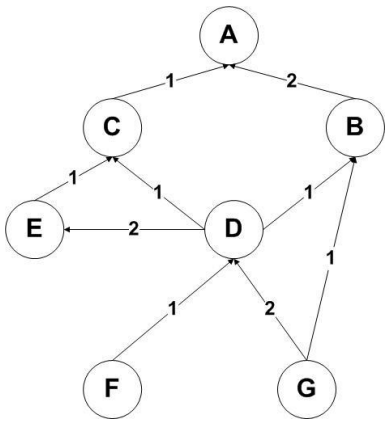
Structured & Unstructured Data

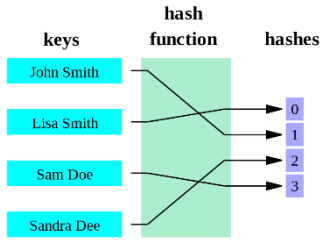
- Relational Databases:
  - Organise data into structured tables and rows
  - Relations have to be simple, they cannot contain any structure such as a nested record or a list
- In memory data structures
  - Much more varied structure
  - Lists, Queues, Stacks, Graphs, Hashing

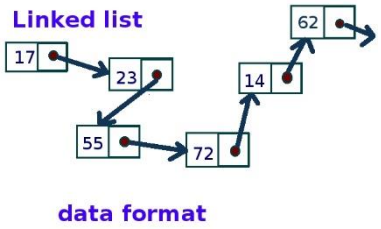
Relational Database



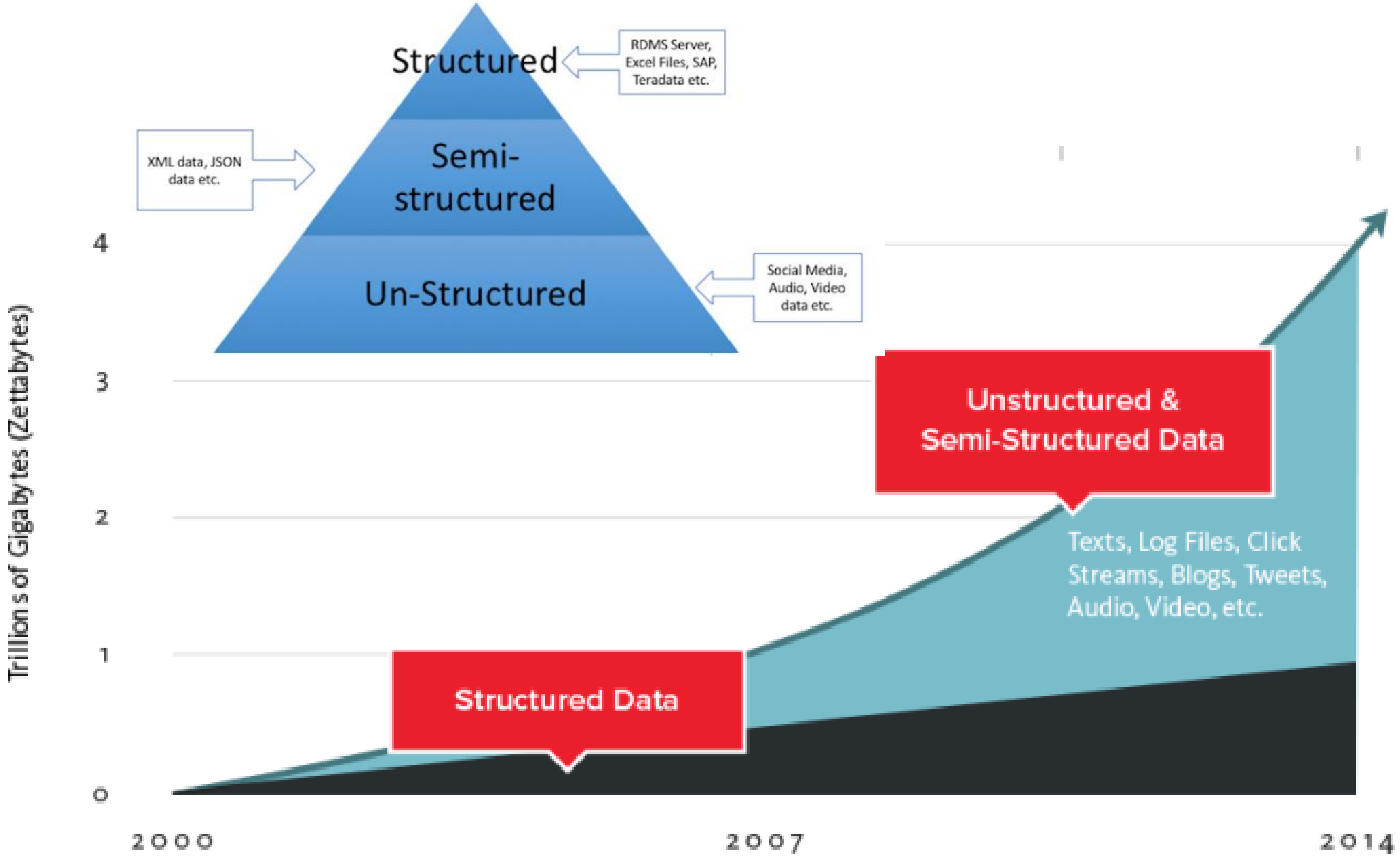
In Memory Data Structures







data format


Structured
RDMS Server, Excel Files, SAP, Teradata etc.
Semi-structured
XML data, JSON data etc.
Un-Structured
Social Media, Audio, Video data etc.
Trillions of Gigabytes (Zettabytes)
4
3
2
1
0
Unstructured & Semi-Structured Data
Texts, Log Files, Click Streams, Blogs, Tweets, Audio, Video, etc.
Structured Data
2000
2007
2014

Databases in the Cloud

- For some apps, a traditional relational database may not be the best fit
  - Organisations are capturing more data and processing it quicker – can be expensive/difficult on traditional DB
  - Traditionally, relational database is designed to run on a single machine in predicable environment
  - May be economic to run large data and computing loads on clusters.
  - Hard to estimate scaling requirements, particularly if it's a web app?
  - Are you going to do Data mining?
- One approach is to use the **Cloud** for you DB
  - Designed for scale
  - Can be outsourced so you don't have to deal with infrastructure requirements.

Cloud DB Advantages

- Removes Management costs
- Inherently scalable
- Latency is predictable and constant
- No need to define schemas(if NoSQL) etc.
- Lots of Cloud DB offerings out there
  - SQL based
  - NoSQL based
- If organisation policy/standards do not allow outsourcing:
  - Can host yourself, most NoSQL DBs are free.

Cloud Database Practices

- Drop Consistency
  - this makes distributed systems much easier to build
- Drop SQL and the relational model
  - simpler structures are easier to distribute:
    - key/value pairs
    - **structured documents**
    - **pseudo-tables**
    - tend to be schema-free, accepting data as-is
- Offer HTTP interfaces using XML or **JSON**
  - Web APIs!!!

Designing Distributed Data

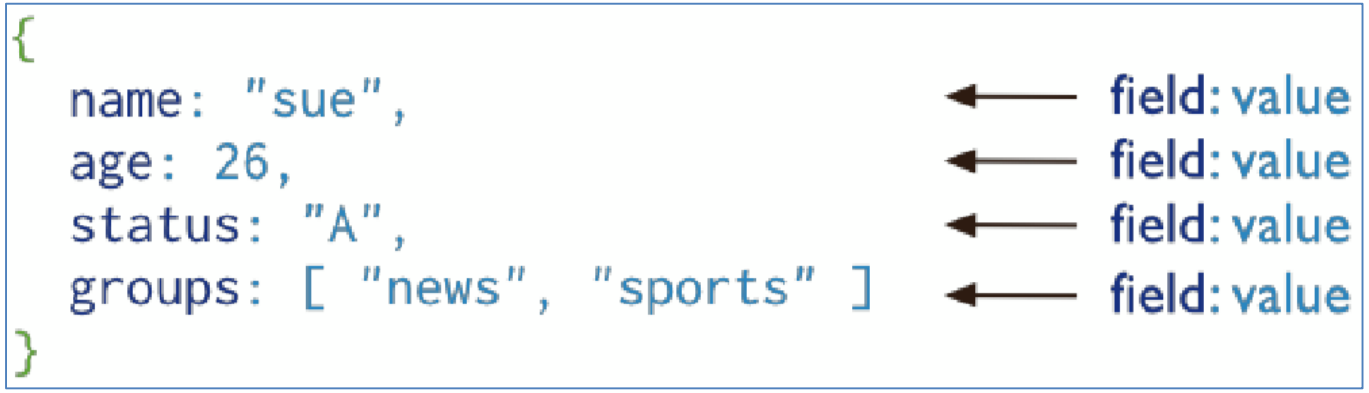
- App data is not homogeneous
  - some kinds of data will be much larger

- consider using different databases for different requirements:
- user details,billing - needs consistency
  - require traditional database
- user data,content - needs partition tolerance
  - replicate to keep safe
- analytics,sessions - needs availability
  - "eventually consistent" is good enough

MONGODB

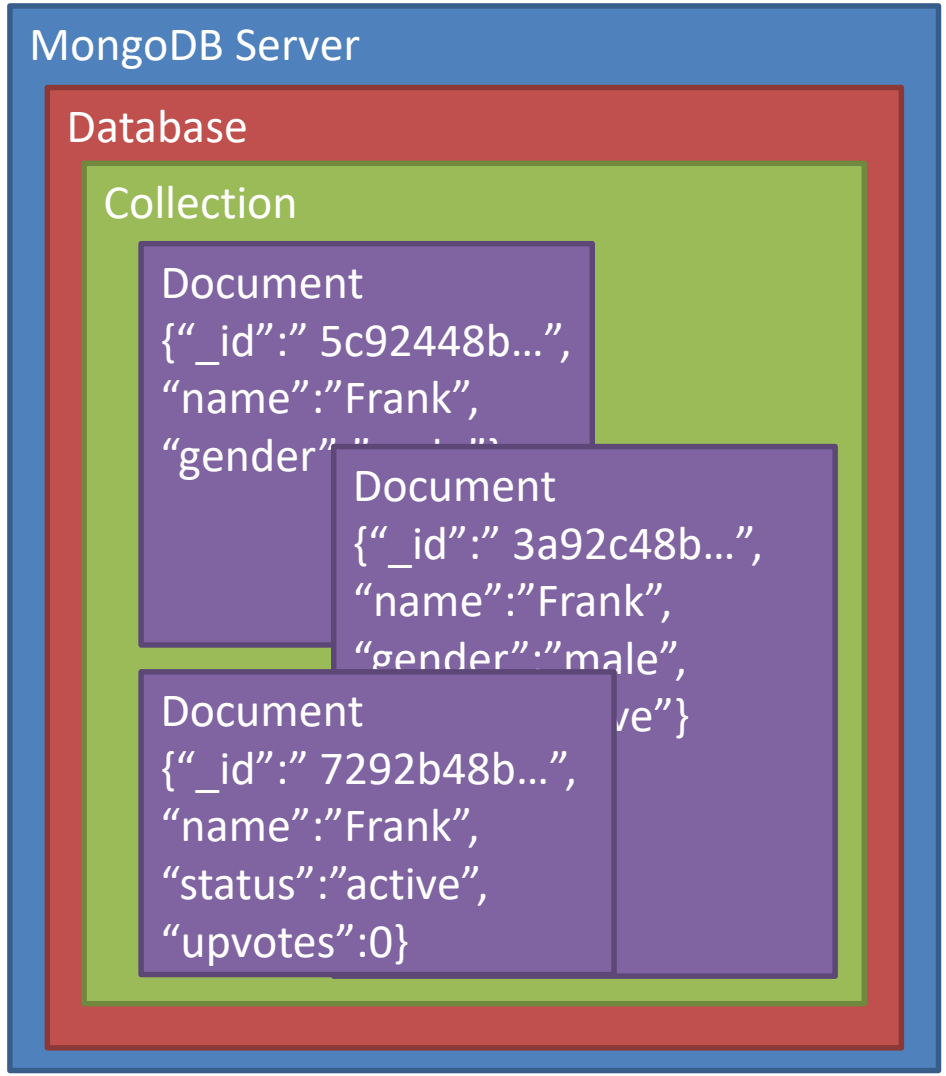
Introduction

- Document-oriented database
- A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- MongoDB documents are similar to JSON objects
- Field Values can be other documents, arrays, arrays of other documents.
  - Reduces need for “Joins”
- Community support - popular choice



Mongo Terminology

- Each **database** contains a set of "Collections"
- Collections contain a set of JSON documents
  - there is no schema (in the DB…)
- The documents can all be different
  - means you have rapid development
  - adding a property is easy - just starting using in your code
- Makes deployment easier and faster
  - roll-back and roll-forward are safe - unused properties are just ignored
- Collections can be indexed and queries
- Operations on individual documents are atomic



Mongo Documents

- MongoDB stores data records as BSON documents.
  - BSON is a binary representation of JSON documents.
- Each document stored in a collection requires a unique _id field and is reserved for use as a primary key.
- If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field.
  - ObjectId values consist of 12 bytes.

```
_id: ObjectId("5c92448b7fbccf28a0c501aa")
name: "Contact 4"
address: "49 Upper Street"
phone_number: "934-4290"
```

Getting Started (locally)

Install Mongo community edition for your OS:

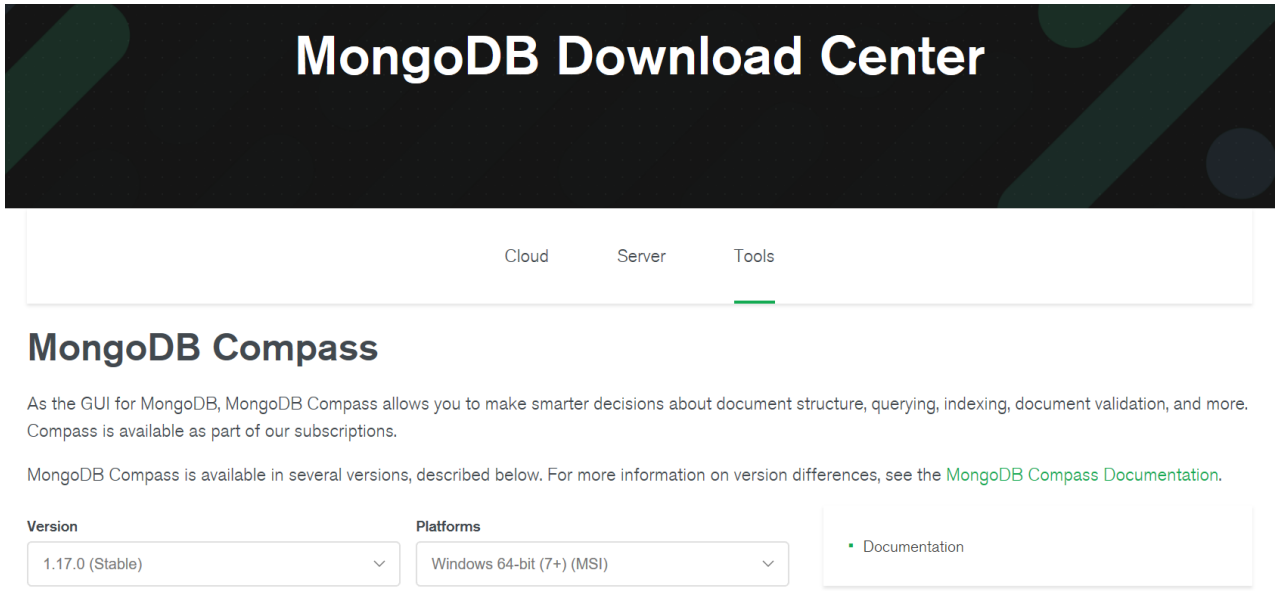
Specify a directory for your db files and start Mongodb server.

```
mkdir db

mongod -dbpath db
```

Getting Started (locally)

- Install Mongo Compass, Graphical User Interface for managing MongoDB.
  - For windows, comes as part of mongodb install
  - Other platforms can get it here:



MONGOOSE

Mongoose Overview

- Mongoose is a object-document model module in Node.js for MongoDB
  - Wraps the functionality of the native MongoDB driver
  - Exposes models to control the records in a doc
  - Supports validation on save
  - Extends the native queries

Mongoose first?

- Shortcut to understanding the basics
- Similar to Object Relational Mapping libraries like JPA/Hibernate
- Easier concept if coming from relational DB background.

Installing & Using Mongoose

1. Run the following from the CMD/Terminal

```
npm install --save mongoose
```

2. Import the module

```
import mongoose from 'mongoose';
```

3. Connect to the database

```
mongoose.connect(process.env.mongoDB);
```

Mongoose Schemas and Models

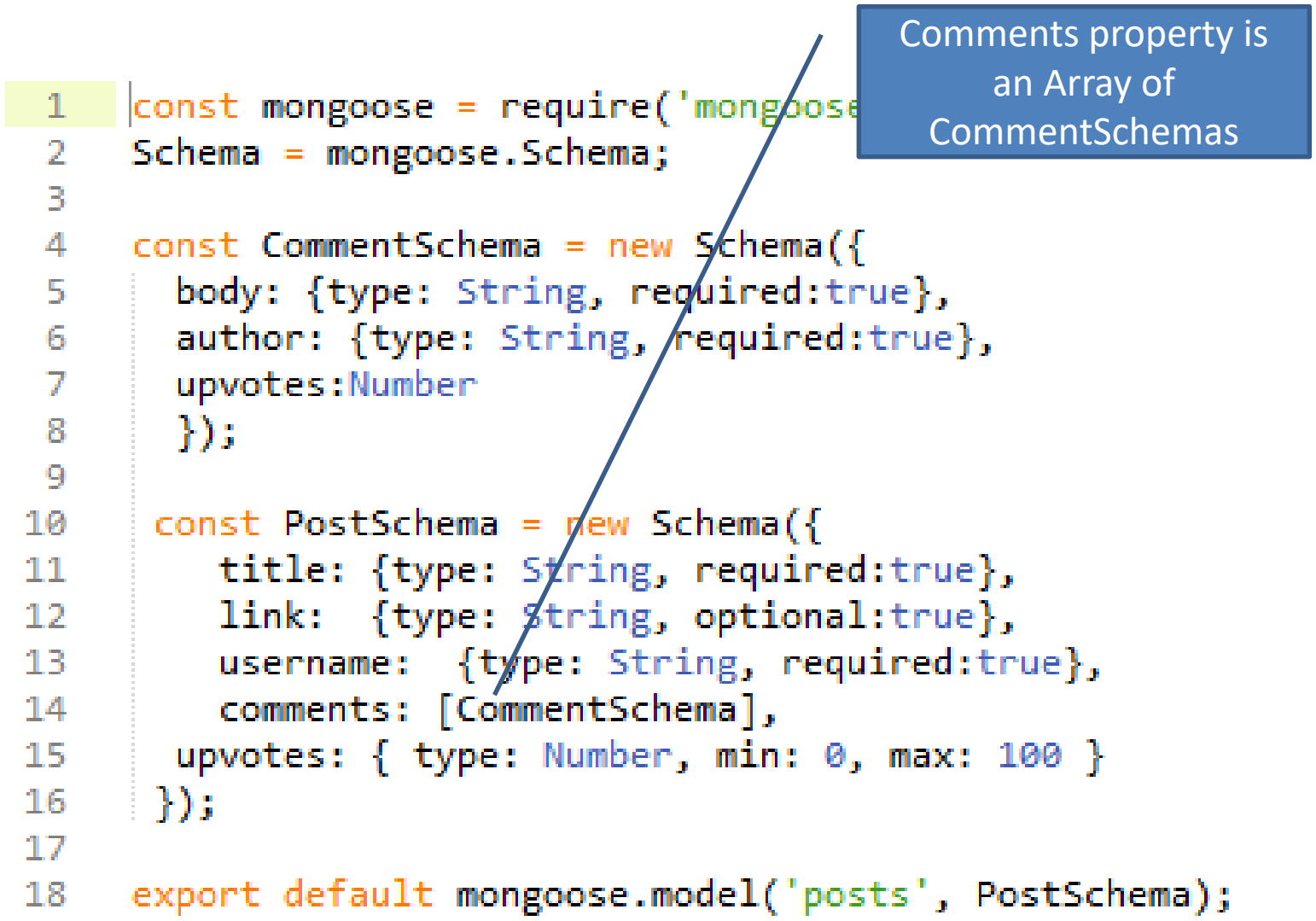
- Mongoose supports models
  - i.e. fixed types of documents
  - Compiled from a **mongoose.Schema definition**
  - Each of the properties must have a type
    - Number, String, Boolean, array, object
- Instances of models represent documents in the Database
- All document manipulation (create/read/update/delete) is handled by models

```
const mongoose = require('mongoose'),
Schema = mongoose.Schema;

const ContactSchema = new Schema({
  name: String,
  address: String,
  age: Number,
  email: String,
  updated: Date
});

const ContactModel = mongoose.model('contacts', ContactSchema);
```

Mongoose Schemas – Arrays & sub-documents



Mongoose Schema – Built-in Validation

- constraints on properties :

```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const ContactSchema = new Schema({
  name: {type: String, required:[true, 'Name is a required property']},
  address: String,
  age: {
    type: Number,
    min: 0,
    max: 120,required: true
  },
  email: String,
  updated: {
    type: Date,
    default: Date.now,
  },
});

export default mongoose.model('Contact', ContactSchema);
```

Mongoose Custom Validation

- Developers can define custom validation on their properties (e.g. validate email field is correct format)

```
ContactSchema.path('email').validate((email) => {
  var emailRegex = /^([\w-\.]+@([\w-]+\.)+[\w-]{2,4})?$/;
  return emailRegex.test(email);
}, 'A valid e-mail address is required');
```

Using Regular Expression (regex) to test for a valid email. If you've not come across them before check out https://www.w3schools.com/jsref/jsref_obj_regexp.asp

Data Manipulation Mongoose

- Mongoose supports all the CRUD operations:
  - Create –> Model.create()
  - Read –> Model.find()
  - Update –> Model.update(condition, props, cb)
  - Remove –> Model.remove()
- Can operate with "*error first*" callbacks or promises.

Create Contact with Mongoose

```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;

const ContactSchema = new Schema({
  name: String,
  address: String,
  age: {
    type: Number,
    min: 0,
    max: 120,
  },
  email: String,
  updated: {
    type: Date,
    default: Date.now,
  },
});

export default mongoose.model('Contact', ContactSchema);
```

```
// Create a contact, using async handler
router.post('/', asyncHandler(async (req, res) => {
  const contact = await Contact.create(req.body);
  res.status(201).json(contact);
}));
```

Update Contact with Mongoose

```
// Update a contact
router.put('/:id', asyncHandler(async (req, res) => {
  if (req.body._id) delete req.body._id;
  const contact = await Contact.update({
    _id: req.params.id,
  }, req.body, {
    upsert: false,
  });
  if (!contact) return res.sendStatus(404);
  return res.json(200, contact);
}));
```

Mongoose Queries

- Mongoose provides a more expressive version of the native MongoDB
  - Instead of:
    {$or: [{conditionOne: true}, {conditionTwo: true}]
  - Do: .where({conditionOne:true}).or({conditionTwo: true})

Mongoose Queries

- Mongoose supports many queries:
  - For equality/non-equality
  - Selection of some properties
  - Sorting
  - Limit & skip
- All queries are executed over the object returned by Model.find*()
  - Model.findOne() returns a single document, the first match
  - Model.find() returns all
  - Model.findById() queries on the _id field.

```
// Delete a contact
router.delete('/:id', asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) return res.send(404);
  await contact.remove();
  return res.status(204).send(contact);
}));
```

Mongoose Queries

- Can build complex queries and execute them later

```
1  const query = ContactModel.where('age').gt(17).lt(66)
2                            .where('county').in(['Waterford','Wexford','Kilkenny']);
3
4  query.exec((err,contacts)=>{...})
5
6
```

- The above finds all contacts where age >17 and <66 and living in either Waterford, Kilkenny or Wexford

Mongoose Sub-Docs

- Ex: Hacker News – Adding a comment to a post.

```
// add comment
router.post('/:id/comments', asyncHandler( async (req, res) => {
   const id = req.params.id;
   const comment = req.body;
   const post = await Post.findById(id);
   post.comments.push(comment);
   await post.save();
   return res.status(201).send({post});
}));
```

Mongoose Sub-Docs

- Updating a Sub-Document(e.g. incrementing the upvotes for a comment)

```
router.post('/:postId/comments/:commentId/upvotes', asyncHandler( async (req, res) => {
   const commentId = req.params.commentId;
   const postId = req.params.postId;
   const post = await Post.findById(postId);
   post.comments.id(commentId).upvotes++;
   await post.save();
   return res.status(201).send({post});
}));
```

Each subdocument is assigned it's own _id from MongoDB. This is a special method to access sub documents

Mongo Sub docs

- Removing a sub document

```
router.delete('/:postId/comments/:commentId', asyncHandler( async (req, res) => {
   const commentId = req.params.commentId;
   const postId = req.params.postId;
   const post = await Post.findById(postId);
   post.comments.id(commentId).remove();
   await post.save();
   return res.status(201).send({post});
}));
```

SCHEMA METHODS

Static & Instance Methods

```
ContactSchema.statics.findByEmail = (email) => {
  return this.findOne({
    email: email
  });
};

ContactSchema.methods.compareEmail = function (candidateEmail) {
  const isMatch = this.email === candidateEmail;
  if (!isMatch) {
    throw new Error('Password mismatch');
  }
  return this;
};
```

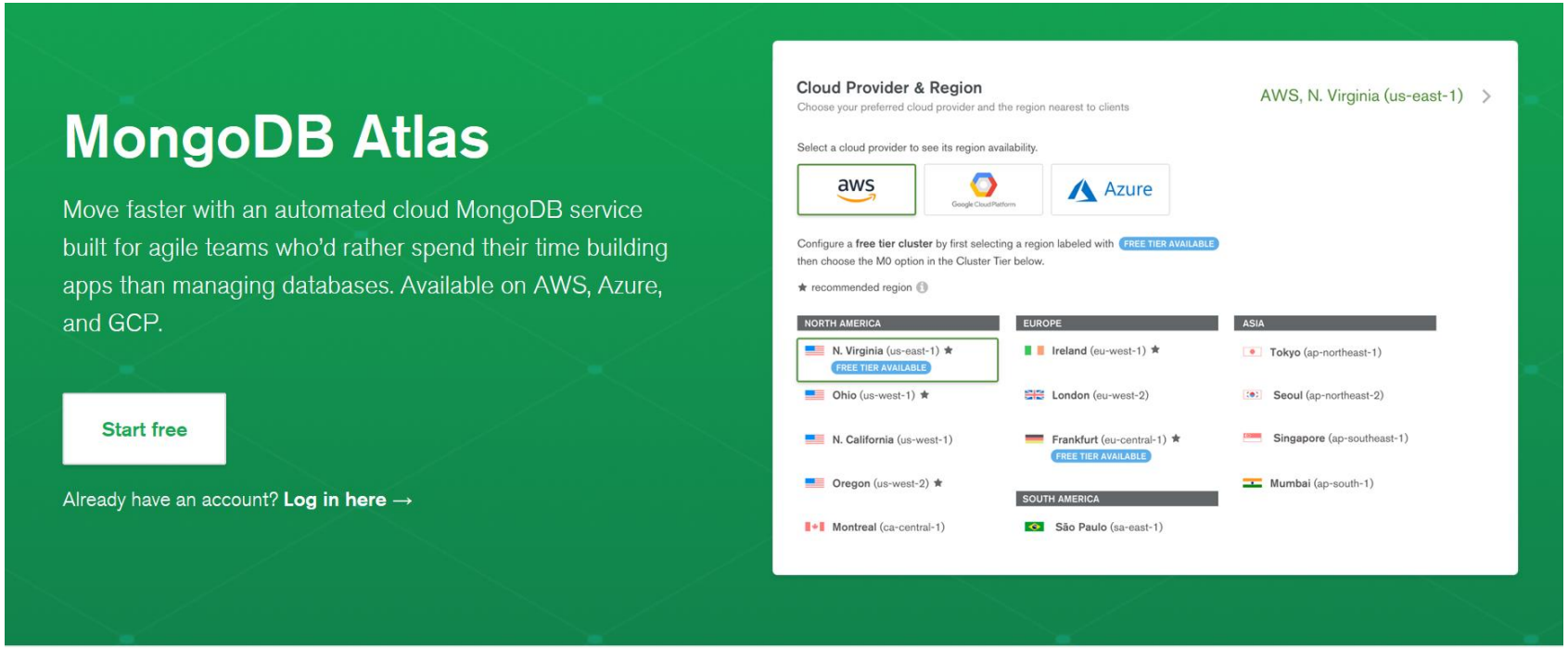
Static Method: belongs to schema. Independent of any document instance

Instance Method: belongs to a specific document instance.

MongoDB as a Service

- Best practice for initial development is to host MongDB process on your development machine
- In production environments, Mongo will be hosted:
  - on it's own instance or
  - provisioned as a service

MongoDB as a Service



MongoDB as a Service

- Some providers allow free access tier
- Provide user credentials wrapped in a URL
- All you need to do is update your config with the relevant URL
- Again, be careful to ignore credentials when pushing to github/public repo